IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 3, No 7, May 2010
ISSN (Online): 1694-0784
ISSN (Print): 1694-0814

A practical application of software security in an undergraduate software engineering course

Cynthia Y. Lester
Computer Science, Tuskegee University
Tuskegee, Alabama 36088, USA

Abstract

Computer software is developed according to software engineering methodologies. However, as more of the economy and our social lives move online, software security has become a topic of increasing importance. Traditionally, courses in software security are offered at the graduate level or in a stand-alone course at the undergraduate level, with many undergraduate students being required to apply security principles to their software development processes as soon as they complete their degrees. Therefore, this paper posits that software security can be effectively introduced to undergraduate students in a traditionally taught software engineering course. The paper presents a modified software engineering course which introduces the secure development life cycle. Several traditional software development methodologies are presented which provide a foundation for introducing secure software principles. Additionally, the paper introduces collaborative learning and service-learning which are used in the practical application of software security concepts. Lastly, challenges and future work are presented.

Keywords: *Collaborative learning, Service-learning, Software development, Software engineering, Software security, Undergraduate students***.**

1. Introduction

Securing information is not a new idea. In fact, securing data has its origins in World War II with the protection and safeguarding of data which resided on mainframes that were used to break codes [1]. However during the early years, security was uncomplicated since the primary threats included physical theft of the system, espionage and sabotage against product resources [1]. Yet, it was not until the early 1970s that the concept of computer security was first studied. With the invention of the Advanced Research Projects Agency Network (ARPANET) by the U.S. Department of Defense in 1968 and its growing popularity in the early 1970s, the chance for misuse increased for what is now known as the modern day Internet. Consequently, with the advent of the Internet and the World Wide Web, protecting data has become a topic of importance.

In the *Report of the Presidential Commission on Critical Infrastructure Protection,* it was stated that "education on methods of reducing vulnerabilities and responding to attacks" and "programs for curriculum development at the undergraduate and graduate levels" were recommended to reduce the number of vulnerabilities and malicious attacks on software systems [2]. Additionally, in the 2003 *National Strategy to Secure Cyberspace* four major actions and initiatives for awareness, education, and training were identified which included [3]:

- Foster adequate training and education programs to support the Nation's cyber security needs
- Promote a comprehensive national awareness program to empower all Americans—businesses, the general workforce, and the general population - to secure their own parts of cyberspace
- Increase the efficiency of existing federal cyber security training programs
- Promote private-sector support for well-coordinated, widely recognized professional cyber security certifications

Therefore, in order to protect data from hackers and saboteurs in a global society where e-commerce, e-business, and e-sharing are the "norm", professionals should have sound knowledge in methods to protect data. Hence, the area of information assurance (IA) has become one of great significance.

Information assurance as defined in the *CNSS Instruction Handbook No. 4009* are measures that protect and defend information and information systems by ensuring their availability, integrity, authentication, confidentiality, and nonrepudiation. Additionally, the measures include providing for restoration of information systems by incorporating protection, detection, and reaction capabilities [4]. In order for students to gain training in information assurance, a series of courses are often taken which include traditional computer science courses but also courses in information security, network security, computer security, cryptography, software security, etc. However, unless an institution has an information assurance track or program, students may not have the

opportunity to gain exposure to many of these concepts, especially those concepts found in a software security course.

Typically, undergraduate software engineering courses do not place an emphasis on security concepts. Consequently, it is difficult for students enrolled in an undergraduate software engineering course to gain training in the secure life cycle. However, undergraduate software engineering courses offer excellent opportunities to introduce the concepts of a trusted development environment and the cost of producing software under such restriction. Further, the identification and discovery of the system security policy and how one models the policy for implementation within a working system provide good opportunities for student exercises.

Therefore, the aim of this paper is to present a modified software engineering course which introduces the secure development life cycle. The paper presents several traditional methodologies and the secure software development life cycle introduced to students. Additionally, the paper introduces the concepts of collaborative learning and service-learning which were used to enhance the practical application of software development and security concepts.

2. Traditional software development

Software engineering is defined as "being concerned with all aspects of the development and evolution of complex systems where software plays a major role. It is therefore concerned with hardware development, policy and process design and system deployment as well as software engineering [5]."

The term software engineering was first proposed at the 1968 NATO Software Engineering Conference held in Garmisch, Germany. The conference discussed the impending software crisis that was a result of the introduction of new computer hardware based on integrated circuits [5]. It was noted that with the introduction of this new hardware, computer systems were becoming more complex which dictated the need for more complex software systems. However, there was no formalized process to build these systems which put the computer industry at jeopardy because systems were often unreliable, difficult to maintain, costly, and inefficient [5]. Consequently, software engineering surfaced to combat the looming software crisis.

Since its inception, there have been many methodologies that have emerged that lead to the production of a software product. The most fundamental activities that are common among all software processes include [5]:

- *Software specification* – the functionality of the system and constraints imposed on system operations are identified and detailed
- *Software design and implementation* – the software is produced according to the specifications
- *Software validation* – the software is checked to ensure that it meets its specifications and provides the level of functionality as required by the user
- *Software evolution* – the software changes to meet the changing needs of the customer

Typically, students are introduced to these activities in the undergraduate computer science curriculum through a software engineering course. This course is sometimes a survey course which exposes students to a variety of life cycle models used in industry. The course is often taught from a systems approach which places an emphasis on creating requirements and then developing a system to meet the requirements. In the traditional view of software development, requirements are seen as the contract between the organization developing the system and the organization needing the system [6].

2.1 The waterfall model

A traditional view of software development is the waterfall method. The waterfall method was the first published software development process and forms the basis for many life cycles. It was noted as a great step forward in software development [7]. The method has stages that cascade from one to the other, giving it the "waterfall" name. Figure 1 is an example of the waterfall life cycle [8].

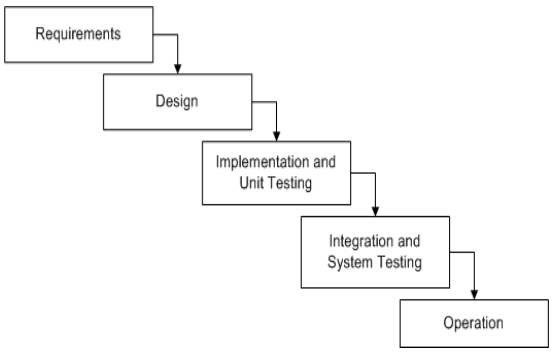

Figure 1. Waterfall model

It has been noted that the method might work satisfactorily if design requirements could be addressed prior to design

creation and if the design were perfect prior to implementation [7]. Consequently, one of the main disadvantages of this model is that requirements may change accordingly to meet the needs of the customer and the change is difficult to incorporate into the life cycle. As a result of this shortcoming, additional life cycles emerged which allowed for a more iterative approach to development.

2.2 Evolutionary development

Evolutionary development is based on the idea of developing an initial implementation and then exposing the build to the user for comment and refinement [5]. Figure 2 is an example of the evolutionary development method [5].

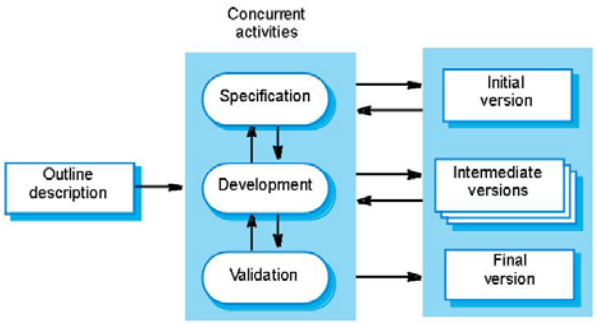

Figure 2. Evolutionary development

There are two fundamental types of evolutionary development:

- *Exploratory development* – developers work with customers to discern requirements and then the final system is delivered
- *Throwaway prototyping* – used to quickly develop a concept and influence the design of the system

The advantage of evolutionary development is that it is developing specifications incrementally [5]. As customers have an opportunity to interact with the prototype, specifications are refined which leads to a better, more useful, usable, and used software. However, while this approach is somewhat better than the waterfall model, it is not without its criticisms. Sommerville notes that the process is not visible and that the systems being developed are often poorly structured [5]. The next model presented is stated to be an improvement over both the waterfall and evolutionary development models.

2.3 Spiral development

The spiral development model is an example of an iterative process model that represents the software process as a set of interleaved activities that allows activities to be evaluated repeatedly. The model was presented by Barry Boehm in his 1988 paper entitled *A Spiral Model of Software Development and Enhancement* [9]. The spiral model is shown in figure 3. The spiral model differs from the waterfall model in one very distinct way because it promotes prototyping; and, it differs from the waterfall and evolutionary development methods because it takes into consideration that something may go wrong which is exercised through risk analysis.

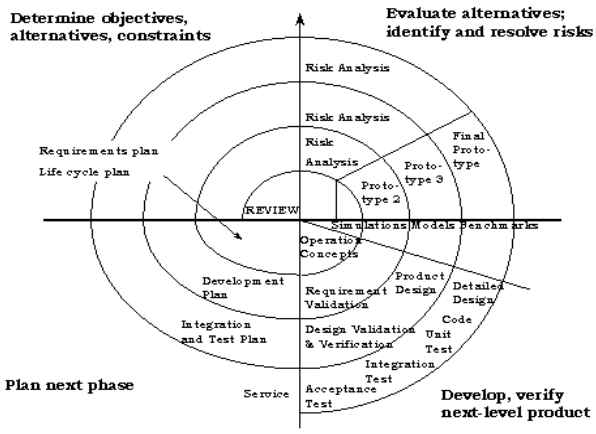

Figure 3. Spiral model

It is noted that this life cycle provides more flexibility than its more traditional predecessors. Further, this method produces a preliminary design. This phase of the life cycle was added specifically in order to identify and resolve all the possible risks in the project development. Therefore, if risks indicate any kind of uncertainty in requirements, prototyping may be used to proceed in order to determine a possible solution.

However, in these approaches as with many of the other approaches to software development that are taught in traditional software engineering courses, security is mostly absent from the process. Hence, the increasingly important need to include a discussion of software security in the software development process taught to undergraduate students. The next section explores secure software development and its life cycle.

3. Characteristics of good software

According to statistics published by the Computer Emergency Response Team (CERT), over 19,500 security vulnerabilities have been reported since 1995. Figure 4 shows the significant increase in vulnerabilities over a ten year period. Similarly, in a report published by CERT in

2005, the number of new security vulnerabilities reported each day almost doubled in comparison to those reported each day in 2004 [10]. It has been reported that these software vulnerabilities and software errors cost the U.S. approximately $59.5 billion annually [11].

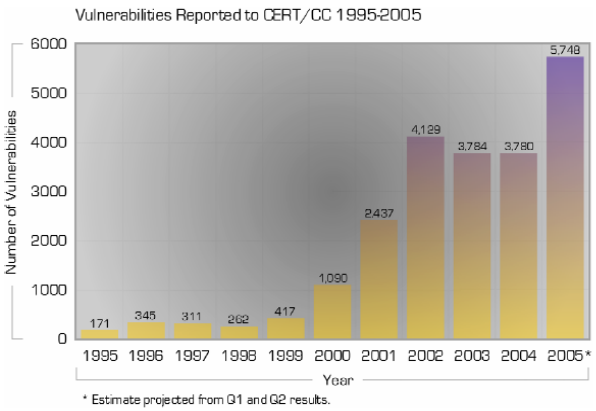

Figure 4. Vulnerabilities Report

Software errors have grown in complexity. In 2000, the National Institute of Standards and Technology (NIST) reported that the total sales of software reached approximately $180 billion and the software was supported by a workforce that consisted of 679,000 software engineers and 585,000 computer programmers [11]. Some of the reasons that software errors have grown in complexity are that typically, software now contains millions of lines of code, instead of thousands; the average product life expectancy has decreased requiring the workforce to meet new demands; there is limited liability among software vendors; and, there is difficulty in defining and measuring software quality [11].

Consequently, it is imperative that students in computer science and information technology be trained in the concepts of security and how to design and develop secure software so that they can contribute viably to the fast changing technological demands of this global society. The traditional development strategies expose students to the methods for software development, but as they consider how to guard against hackers, how to protect critical information, and how to lessen security threats, a question of what is “good” information arises. Therefore, before students can understand and have an appreciation of the secure software development life cycle, they must first be exposed to the qualities and characteristics of “good” information.

The value of information has been stated to come from the characteristics that it possesses [1]. While some characteristics may increase the value of the information as it relates to use by users, other characteristics may have a more significant value among security professionals. However, all characteristics as defined below are critical as it relates to secure information [1].

3.1 Availability

Availability allows users who need to access information to access the information without impediment or intrusion. Further, availability means that users can receive information in the desired format.

3.2 Accuracy

Accuracy as defined by *The American Heritage College Dictionary* is conformity to fact; precision; exactness [12]. As accuracy relates to secure software it means that the software has the value that the user expects and that it is also free from errors.

3.3 Authenticity

Authenticity is the state or quality of information being original or genuine. The information should not be a replication of other information. Whitman further reveals that information is authentic when it is the information that was originally created, placed, stored or transferred [1].

3.4 Confidentiality

Confidentiality of information is another characteristic of “good” information. While this concept is not difficult for students to understand and is one that comes to most minds of students, it is still important that students understand that confidentiality is more than protecting information. Confidentiality of information means that only those persons with “certain” rights can have access to the information. It means that only authorized persons or systems can gain access to the information.

3.5 Integrity

Another characteristic of information is that of integrity. Integrity is adherence to a strict code or the state of being unimpaired [12]. As it relates to the integrity of information it is the state of being uncorrupted or the state of being whole. It is important to point out to students that corruption of information does not necessarily happen as a result of saboteurs, but also as data moves from one system to another.

3.6 Utility

The sixth characteristic of information is that of utility. Utility is the condition of being useful. While students understand that information must be available, authentic, and accurate, if the information being provided is not

useful or presented in a format that cannot be used, then the information loses its value or its quality of being "good" information.

3.7 Possession

The final characteristic of information as stated and defined by Whitman and Mattford is that of possession. Possession is the condition of being owned or controlled. Whitman and Mattord note that while a breach in confidentiality always results in a breach of possession, the opposite may not be true. Consequently, students should understand the role that possession plays in the characteristics of quality information [1].

4. The secure development life cycle

There are many approaches to the development of robust software that can ensure that the information being used by users is available, accurate, authentic, possesses confidentially, integrity, is useful and can be controlled. However, the question becomes how to introduce this model at the undergraduate level when a specialized course in software security is not available or when typically students only take one course in software development. The following section presents the secure software development life cycle taught in a traditionally taught software engineering course and introduces a method which allows students to gain practical experience in implementing security concepts.

A misconception among students as well as with computing professionals is that security should be thought of in the later phases of the software development life cycle. However, if systems are to withstand malicious attack, a robust software development model or a secure software development method must be used. Figure 5 presents one viewpoint of the secure life cycle discussed in class [13].

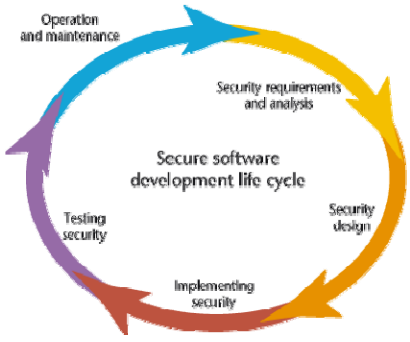

Figure 5. Secure life cycle

4.1 Security requirements and analysis

While requirements are being gathered from users and stakeholders, focus should also be placed on establishing a security policy. In order to develop a security policy, attention needs to be given to what needs to be protected, from whom, and for how long [14]. Additionally, thought needs to be placed on the cost of protecting the information. The result of this phase should be a set of guidelines that create a framework for security [1].

4.2 Security design

During the design phase it has been stated that the security technology needed to support the framework outlined in the requirements phase is evaluated, alternative solutions are explored, and a final design is agreed upon [1]. It is recommended by Viega and McGraw that the following be the focus of this phase [14]:

- How data flows between components
- Users, roles and rights that are explicitly stated or implicitly included
- The trust relationships between components
- Solutions that can be applied to any recognized problem

At the end of this phase a design should be finalized and presented. The design should be one that can be implemented.

4.3 Implementation

The implementation phase in the secure development life cycle is similar to that which is found in traditional methodologies. However, when implementing a software project with security in mind, it is important to consider a language or a set of languages that may have security features embedded, one that is reliable when it comes to denial-of-service attacks, and that can perform error checking statically, etc. Further, it is important to understand the weaknesses of languages, for example buffer overflows in C and C++.

4.4 Testing

Testing in the secure development life cycle is different than in traditional methodologies. In traditional methodologies, testing is done to ascertain the behavior of the system and to determine if the system meets the specifications. Security testing is used to determine if a system protects data and maintains functionality as intended. As mentioned previously the six concepts that need to be covered by security testing are availability, accuracy, authenticity, confidentiality, integrity, utility, and possession. It has been stated that security testing is most effective when system risks are uncovered during

analysis, and more specifically during architectural-level risk analysis [14].

4.5 Maintenance

It has been stated that the maintenance and change phase may be the most important phase of the secure development life cycle given the high level of cleverness seen in today's threats [1]. In order to keep up with the changing threats to systems, security systems need constant updating, modifying, and testing. Constant maintenance and change ensure that systems are ready to handle and defend against threats.

5. The modified software engineering course

The next sections describe the modified software engineering course, *CSCI 430-Software Engineering*. The course was revised to include software security concepts both in lecture materials and the semester long service-learning project students were to complete. Learning resources may be of the following two broad types: (a) lecture materials; and (b) modification of existing syllabi and project descriptions/laboratory exercises. Lecture materials introduce specific issues, concepts, terminology, and techniques to the students in the classroom. Further the course introduced to students the concepts of collaborative learning and service-learning, which are presented in the next sections, as well as the course description.

5.1 Collaborative learning

Students learn best when they are actively involved and engaged in the learning process. In educational environments, study groups are often formed to gain better insight on course topics through collaborative efforts. Collaborative learning is defined as the grouping and/or pairing of students for the purpose of achieving an academic goal [15]. Davis reported that regardless of the subject matter, students working in small groups tend to learn more of what is taught and retain it longer, than when the same content is presented in other more traditional instructional formats [16].

Supporters of collaborative learning suggest that the active exchange of ideas within small groups not only increases interest among group participants, but also helps to improve critical thinking skills. The shared learning environment allows students to engage in discussion, take responsibility for their own learning, hence becoming critical thinkers [15]. Students are responsible for their own learning as well as the learning of others. Research has shown that collaborative learning encourages the use of high-level cognitive strategies, critical thinking, and positive attitudes toward learning [17]. Further, Johnson and Johnson suggest that collaborative learning has a positive influence on academic performance [18].

Informal learning groups. Informal learning groups are ad hoc temporary grouping of students within a single class period. Informal groups can be organized at any time during class and are often used to monitor students' understanding of material, to provide students an opportunity to implement what is being discussed or to change the pace of class discussion.

Study teams. Study teams are long-term groups with stable membership whose primary responsibility is to provide members with encouragement, support, and/or assistance in completing course requirements and assignments.

Formal learning groups. Formal learning groups are teams established to complete a specific task. These groups may complete their work in a single class period or over the course of several weeks. Students tend to work together until the task is finished, and their project is graded.

The modified course made use of formal learning groups.

5.2 Service-learning

Research has found that of the females and the minority students who choose computer science as a major, many place an intrinsic value on how technology can "better the world and its people" and value the role that computers play in the future [19]. Consequently, the idea of service-learning is appealing because it allows these students to see first hand how technology can improve processes and the way of life for citizens in the global society.

Service-learning is defined as a method of teaching through which students apply their academic skills and knowledge to address real-life needs in their own communities [19]. Service-learning provides a compelling reason for students to learn; it teaches the skills of civic participation and develops an ethic of service and civic responsibility. By solving real problems and addressing real needs, students learn to apply classroom learning to real world situations [19]. Service-learning has been shown to be an educational technique that facilitates a student's growth in academics, social maturity, critical thinking, communication, collaboration, and leadership skills [19]. Students who are involved in meaningful service-learning have further been shown to perform better on tests, show a sense of self-esteem and purpose, connect

with the community, and want to be more civically engaged than students who do not participate in service-learning activities [19].

There are many key components that are encompassed within service-learning. The author has chosen some of the activities that were included in the redesigned *CSCI 430* and, they are presented in the next sections.

Reflection. Reflection fosters the development of critical thinking in students. Reflection and critical thinking (problem-solving) are essential tools that will help students be successful in school, career, and life. Service-learning reflection includes the following activities by the student:

- Assessing personal interests, knowledge, skills, and attributes that will be useful in performing the service-learning project.
- Thinking about how to take effective steps to meet the identified needs.
- Self-evaluating one's progress toward meeting the goals of the project.

Working as a team. The students learn to work for a common goal and by doing so acquire a variety of skills, such as how to lead, how to be accountable, how to communicate ideas, how to listen to others, and how to set a goal and work effectively as a team to reach the goal.

Experiential learning. Service-learning uses direct experience and hands-on learning to help the student learn to take the initiative, assume responsibility, and develop effective problem-solving skills.

It was the anticipation of the author that through the hands-on experience of developing a project that included security concepts, students would gain an understanding of the importance of secure software engineering and their approach to development would be enhanced. Further, as students use and understood the concepts presented during class, conceptually they would be able to apply the principles to a semester long service-learning project.

5.3 Course description

A brief description of *CSCI 430 - Software Engineering*, is to provide students with an engineering approach to software development and design; and, to expose students to current research topics within the field [20]. The software engineering course was modified to reinforce the need to think about security features and requirements early in the development process so that security protection mechanisms are designed and built into the system rather than added on at a later time.

The prerequisites for the course are to have successfully completed CSCI 230 (Data Structures) and CSCI 300 (Discrete Mathematical Structures) with a grade of C or better.

Learning outcomes are extremely important when developing a course. The learning outcomes describe the specific knowledge and skills that students are expected to acquire. The learning outcomes for the *CSCI 430* course include the following: at the end of the course, a student should be able to:

- Describe in detail the software process
- Identify various software process models and determine which model should be used for a specific project
- Implement each phase of the software process
- Work effectively and efficiently in a team environment to produce a large scale project
- Identify and discuss current research topics related to the software engineering discipline

To meet the objectives of the learning outcomes, Table 1 presents an outline of the topics covered during the sixteen week semester [20].

Table 1. Weekly course schedule

WEEK	TOPIC
1	Introduction to Software engineering
2	Traditional software processes
3	Traditional software processes
4	Security development life cycle model
5	Software requirements
6	Requirements engineering
7	Managing software security risk
8	Midterm
9	Software security in design
10	Architectural design
11	Object-oriented design
12	User-interface design
13	Verification & validation
14	Software testing
15	Software testing
16	Project implementation

Students were assessed through homework, a semester long service-learning project, and a paper in special topics. Additionally, two exams are administered. The next section presents the service-learning project.

6. Service-learning project

6.1 Problem statement

The semester long project selected for the fall 2009 semester was to develop an electronic voting/tallying system for the hotly contested position of the University's Queen. During past years, there have been errors in the selection process of the University's Queen which has resulted in a process where contestants and the student body have little confidence. Students were required to develop a software product that meets the needs of the customer and helps to refine the election process and ballot-counting process for the University's Queen contest. The selection of this project was appealing because students would be able to see how the development of this technology could improve the process and its impact on contestants, the student body, and administrators.

Students were part of a formal learning group/team which was expected to meet with the customer (or representative) so that each phase of the process could be implemented. The team was also expected to produce a deliverable by the set deadline for each phase of the process and to also deliver it and make presentations to the customer (or representative).

6.2 Learning outcomes

The learning outcomes of the semester long service-learning project stated that after completion the students would:

- Have a working knowledge of the secure software development life cycle
- Understand and have a working knowledge of secure software engineering principles
- Be able to describe software vulnerabilities
- Develop and execute security measures
- Work effectively and efficiently in a team environment to produce the semester long project

6.3 Project requirements

Students were given basic requirements by the author for the software application; however, the majority of the requirements were gathered from stakeholders. Since the project was infused with software security concepts there were both standard project requirements as well as security requirements.

6.4 Project deliverables

Each item that the student team submitted was considered a deliverable. The project had four deliverables which were the requirements document, design document, implementation, and the test plan. The following is an overview of the project deliverables.

Requirements Document. The first document students were required to submit was the requirements document. The requirements document was considered the official statement of what the students would implement. It included both the stakeholder requirements for the software application, which students named the *MISS System*, and a detailed specification of system requirements. To gather the requirements, students met with stakeholders who included Administrators in the Office of Student Life, contestants from past elections, and student body leaders who were in charge of election results. The initial document was meant to get the students active in the planning and development of the system. After completion of the requirements document, students had an idea of the way they wanted the system to look, how the system would be accessed, and by whom.

Design Document. The team was required to use one of the decomposition styles discussed in the course. The design document was required to have an introduction, an overview of the design strategy chosen, and the diagrams, charts, and/or details required as part of the decomposition strategy chosen. The design document was also meant to be an in-depth description of the system design. The design showed how data flowed between system components and the trust relationships between components. Both the system and security requirements were described and explained how they would be implemented. Further the document identified vulnerabilities to the system and possible solutions were presented.

Implementation. Students were required to implement the project based on the requirements and design documents. To implement the project students chose the Java programming language.

Testing. This was probably the most difficult part of the life cycle for students along with the implementation. Students were required to develop a test plan which required them to perform functional testing and security testing. Further, the system would undergo acceptance testing.

To ensure that students stayed on task during the sixteen week semester, a deliverable timeline was created. Table 2 presents the deliverable timeline for the project [20].

Table 2. Deliverable timeline

Deliverable	Deadline
Requirements document	Week 8
Design document	Week 12
Implementation	Week 16
Test Plan	Week 16

7. Results

A version of the course was piloted during the fall 2007 semester. During the fall 2008 semester the course was once again revised in terms of content and project deliverables. The modified course presented in the paper was offered during the fall 2009 semester and encompassed the change in content and deliverables, with the addition of collaborative learning and service-learning. The results of the study are as follows:

- The formal learning group/team created complete deliverables by specified due dates.
- Rotation of team leaders proved helpful in encouraging leadership skills.
- Soft skills of personal communication, collaboration, and interactivity were practiced.

Students were asked to conduct an exit interview after the completion of the course and the service-learning project. The exit interviews revealed the following:

- Students left the course with an appreciation for the software engineering discipline and the secure software development process.
- Students understood the need for documentation throughout the development life cycle.
- Students stated that developing software specifications proved to be extremely challenging and stakeholders are not always sure about the product they want developed.
- Students also stated that performing *all* phases of life cycle proved to be time consuming.

When asked about the course itself, the students expressed the following:

- They appreciated the opportunity to be engaged in a service-learning project.
- Although difficult, they enjoyed creating a software application for "real-customers."
- They enjoyed the collaborative learning experience and desired more courses that infused collaborative learning into the learning experience.

8. Challenges and future work

Developing a course that exposes students to an innovative way of learning proved to be quite challenging. This section describes some of the challenges that the author encountered as well as suggestions for change.

One of the challenges that the author faced was selecting a course project for students that could be completed in one semester. Although the students liked the idea of the service-learning project, they expressed certain levels of discomfort. While the students were very sure of their computing ability, they were less confident with their technical writing skills, and with their ability to ascertain correctly system specifications from their "customers."

An additional challenge was getting students to understand that software engineering is more than just programming. The service-learning project was introduced to students at the beginning of the semester; however, the team waited to start the project thinking that "coding" the software application would not be time consuming. While this assumption might have been true if "coding" was the only part required in the project, once students understood that the specifications had to be gathered from customers, they expressed feelings of hesitation about the service-learning project.

The exit interviews also revealed some concerns by the students, which the author has noted for the next course offering which include:

- To choose a service-learning project in which the development life cycle can be completed in one sixteen week semester. Although the project was introduced early and students and the author thought that the project could be completed, there was still not enough time left in the semester to complete the software development life cycle (i.e., testing was not complete).
- To have a better method for choosing team leaders. Students expressed concerns about the method for choosing team leaders for various parts of the project (i.e., strengths and weaknesses should be taken into account).
- To provide a mechanism for rewarding active team members. Students often expressed their disappointment with classmates who did not actively participate in team meetings, etc.

9. Conclusion

In conclusion, the aim of this paper was to present a modified software engineering course which introduces the secure development life cycle to undergraduate students. The author acknowledges that while there are many development methodologies that exist to train students in software security, many consist of steps that cannot be covered and implemented in one course especially with undergraduate students.

As software becomes more complex and vulnerabilities and threats to these systems become just as complex, it is important to introduce to the next generation of technologist ways that systems can be made more secure. As educators it becomes our responsibility to train these students so that developing secure software is not just introduced in theory, but in practice as well.

Acknowledgments

The author thanks the students enrolled in the *CSCI 430-Software Engineering* course, fall 2009 semester, more specifically J. Brazelton, T. Carroll, and E. Mike for their invaluable contribution to the project.

References

[1] M.E. Whitman and H.J. Mattford. (2004). Principles of Information Security. Boston: Course Technology.

[2] J. Elli, D. Fisher, T. Longstaff, L. Pesante, and R. Pethia. "A Report to the President's Commission on Critical Infrastructure Protection." [Electronic Version] http://www.cert.org/pres_comm/cert.rpcci.ex.sum.html#edu (Accessed on April 1, 2008).

[3] The National Strategy to Secure Cyberspace. (2003). [Electronic Version]. http://www.uscert.gov/reading_room/cyberspace_strategy.pdf (Accessed on November 7, 2007).

[4] http://www.cnss.gov/Assets/pdf/cnssi_4009.pdf.

[5] I. Sommerville. (2007). *Software Engineering 8th Ed.* Addison Wesley, 978-0-321-31379-9, Boston, MA.

[6] C. Angelov, R.V.N. Melnik, & J. Buur. (2003). The synergistic integration of mathematics, software engineering, and user-centered design: exploring new trends in education. *Future Generation Computer Systems.* Vol. 19, 299 – 1307.

[7] B. K. Jayaswal and P.C. Patton. (2007). Design for trustworthy software: Tools, techniques for developing robust software. Prentice Hall, 0-13-187250-8, Upper Saddle Rover, NJ.

[8] Codebetter.com http://codebetter.com/blogs/raymond.lewallen/downloads/waterfalllModel.gif. (Accessed on October 10, 2009).

[9] B. Boehm. (1988). A Spiral Model of Software Development and Enhancement. *IEEE Computer 21*, 5, 61-72.

[10] CERT Coordination Center, CERT/CC statistics 1998 – 2005. [Electronic Version]. http://www.cert.org/stats/cert_stats.html (Assessed January 17, 2008)

[11] Software Errors Cost U.S. Economy $59.5 Billion Annually: NIST Assesses Technical Needs of Industry to Improve Software-Testing [Electronic Version] http://www.nist.gov/public_affairs/releases/n02-10.htm (Accessed on April 1, 2008).

[12] Accuracy; Integrity. American Heritage College Dictionary. (1993). New York: Houghton Mifflin Company.

[13] A. Apvrille and M. Purzandi. "Secure Software Development by Example," IEEE Security & Privacy, vol. 3, no. 4, July/August, 2005. p. 10 – 17.

[14] J. Viega and G. McGraw. (2001). *Building Secure Software: How to Avoid Security Problems the Right Way.* Boston: Addison-Wesley.

[15] Gokhale, A. "Collaborative learning enhances critical thinking." *Journal of Technology Education* 7, no. 1. 1995.

[16] Davis, B.G. Tools for Teaching. San Francisco: Jossey-Bass Publishers. 1993.

[17] Wang, S. and S. Lin. The effects of group composition of self-efficacy and collective efficacy on computer-supported collaborative learning. *Computer and Human Behavior*. 2006.

[18] Johnson, R. T and D.W. Johnson. "An Overview of collaborative learning." *Creativity and Collaborative Learning*; Baltimore: Brookes Press. 1994. [Electronic Version]. http://www.cooperation.org/pages/overviewpaper.html (Accessed on August 31, 2006).

[19] McPherson, K. Service Learning. New Horizons for Learning. http://www.newhorizons.org/strategies/service_learning/front_service.htm. 2005.

[20] C. Lester. *CSCI 430 – Software Engineering Syllabus.* 2009.

Dr. Cynthia Lester earned the B.S. degree in Computer Science from Prairie View A&M University, Prairie View, Texas (1994) and both the M.S. (1996) and Ph.D. (2004) degrees from The University of Alabama, Tuscaloosa, Alabama. She joined the Tuskegee University, Tuskegee, Alabama, faculty in 2005 in her current rank of Assistant Professor of Computer Science. She has held positions as an Instructor of Computer Science at Tuskegee University and at IBM as a Technical Sales Representative. Dr. Lester's areas of specialization are Human Computer Interaction and Software Engineering. She has authored more ten peer-reviewed publications, two magazine articles which appear in the IANewsletter, and one book chapter. Her current work in human-computer interaction focuses on training and educating undergraduate computer science majors in user-centered design. For her work in this area, she garnered the Best Paper Award at the 2008 International Conference on Advances in Computer Human Interaction. Most recently, Dr. Lester was named as a 2010 International Academy, Research, and Industry Association Fellow. She is a member of Sigma Xi, The Scientific Research Society.